

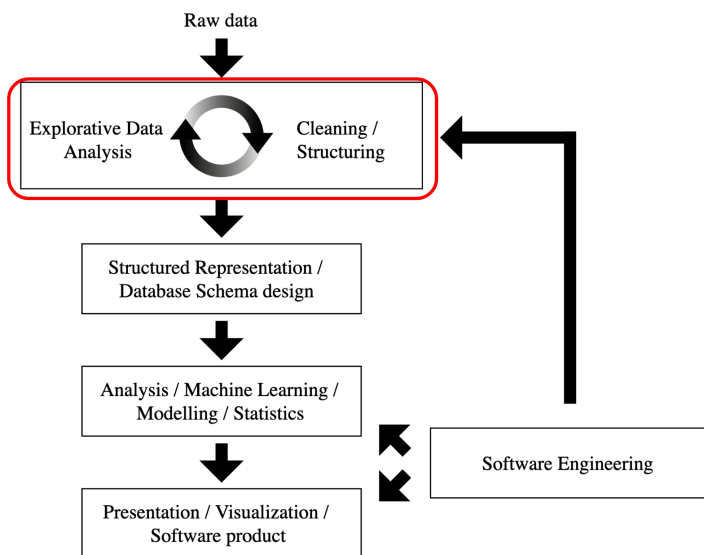
# Individual Pass/Fail Exercise 1 S

[Start Assignment](#)

- Due Friday by 16:00
- Points 0
- Submitting a file upload
- Available 10 Feb at 8:00 - 21 Feb at 17:00

In this assignment you will focus on structuring data using regular expressions, and using regular expressions to clean real text data, analysing the resulting dataset, and plot the associated frequency distributions of words in the dataset.

In terms of the Data Science Pipeline, you will mainly be focusing on data cleaning and explorative data analysis.



## Submission Requirements

1. Jupyter Notebook (.ipynb file) implementing the assignment.
2. PDF printout of the executed Jupyter Notebook displaying the results of your work. (Note that the PDF printout may differ from the notebook, i.e. it may be 200 pages long, so please inspect it are avoid submitting unnecessary content before submission.)

## Part 1: Regular expression warmup

In these questions, you are expected to use functionality from the Python regular expression module: `re`.

- a. Danish CPR numbers consist of a six-digit date (DDMMYY), followed by a 4 digit identifier (IIII). Using the `re` module, the task is to write and apply a regular expression that matches any CPR number, either in the DDMMYYIIII format, or in the DDMMYY-IIII format. The regular expression should contain four groups, such that the DD, MM, YY, and IIII parts can be extracted after matching.

- b. The 4-digit identifier together with the last two digits of the date (i.e. the year), encodes in which century a person is born, using the following system


IIII	YY	Born in century
0001-3999	00-99	1900
4000-4999	00-36	2000
4000-4999	37-99	1900
5000-8999	00-57	2000
5000-8999	58-99	1800
9000-9999	00-36	2000
9000-9999	37-99	1900

Note that in this table, both numbers in a range are included (e.g. 00-99 includes both 00 and 99).

Write a function that returns the relevant century based on the information in the table above, and returns either `1800`, `1900` or `2000` (as an integer).

## Part 2: Processing the FakeNewsCorpus data set

This part focuses on getting the FakeNewsCorpus dataset in a reasonable shape, and doing some initial exploration of the data. You will be working on a sample of the FakeNewsCorpus.

- Download the CSV onto your computer from  
[https://raw.githubusercontent.com/several27/FakeNewsCorpus/master/news\\_sample.csv](https://raw.githubusercontent.com/several27/FakeNewsCorpus/master/news_sample.csv)  
 [https://raw.githubusercontent.com/several27/FakeNewsCorpus/master/news\\_sample.csv](https://raw.githubusercontent.com/several27/FakeNewsCorpus/master/news_sample.csv).
- Read the CSV file into memory
- Manually inspect the data to get an idea of potential problems of the data structure and representation that need to be fixed.
- Clean the data. First, you'll try to do this manually, by writing our own `clean_text()` function that uses regular expressions. The function should take raw text as input and return a version of the text with the following modifications:
  - all words must be lowercased
  - it should not contain multiple white spaces, tabs, or new lines
  - numbers, dates, emails, and URLs should be replaced by "<NUM>", "<DATE>", "<EMAIL>" AND "<URL>", respectively. Note that replacing dates with <DATE> is particularly tricky as dates can be expressed in many forms. You may just choose one or a few common date formats present in the data set and only replace those. (Be careful about tokenizing <> symbols because these are punctuation in most Tokenizers).

5. Now, try to use a library for cleaning the data. The clean-text module

(<https://pypi.org/project/clean-text/> (<https://pypi.org/project/clean-text/>)) provides out-of-the-box functionality for much of the cleaning you did in the previous step (`pip install clean-text`). Use it to implement the same cleaning steps as in your own `clean_text()` implementation.

## Part 3: Descriptive frequency analysis of the data

Now, you will do some simple exploration of the data to understand the effect of preprocessing conducted in Part 2, namely to the vocabulary, and the frequency distribution of the vocabulary across documents.

1. Calculate the number of unique words in the data after preprocessing and compare it to the number of unique words before preprocessing (i.e. in the raw data).
2. Calculate how frequently each of these words is used in the dataset. Sort this list, so that the most frequent word appears first, and then use Python's matplotlib library to plot a barplot where the 50 most frequent words appear on the x-axis, and their frequency appears on the y-axis.